Introduction to Data Management
CSE 344

Lecture 24: MapReduce

HW8 is out

- Last assignment!
  - Get Amazon credits now (see instructions)
- Spark with Hadoop
- Due next wed

Parallel Data Processing @ 1990

Review: Shared Nothing

- Cluster of machines on high-speed network
- Called "clusters" or "blade servers”
- Each machine has its own memory and disk: lowest contention.

NOTE: Because all machines today have many cores and many disks, then shared-nothing systems typically run many "nodes” on a single physical machine.

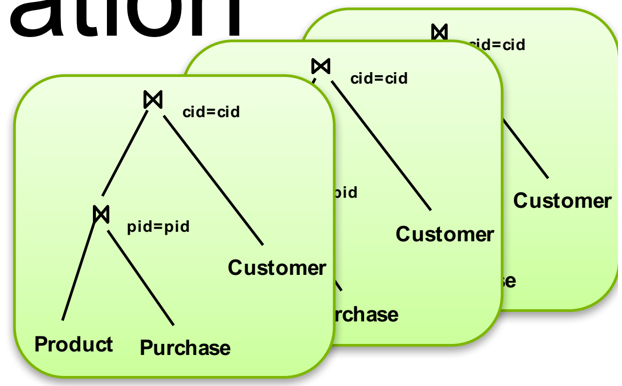
Characteristics:

- Today, this is the most scalable architecture.
- Most difficult to administer and tune.

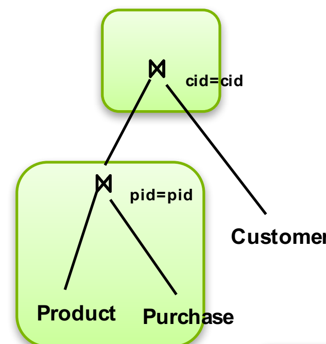
We discuss only Shared Nothing in class

Review: Approaches to Parallel Query Evaluation

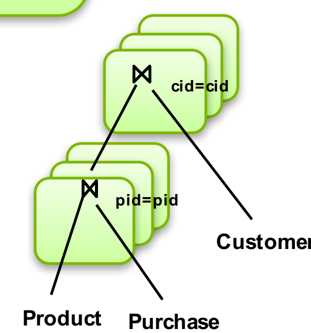
- Inter-query parallelism
  - Transaction per node
  - OLTP



- Inter-operator parallelism
  - Operator per node
  - Both OLTP and Decision Support



- Intra-operator parallelism
  - Operator on multiple nodes
  - Decision Support



We study only intra-operator parallelism: most scalable

Distributed Query Processing

- Data is horizontally partitioned on many servers

- Operators may require data reshuffling

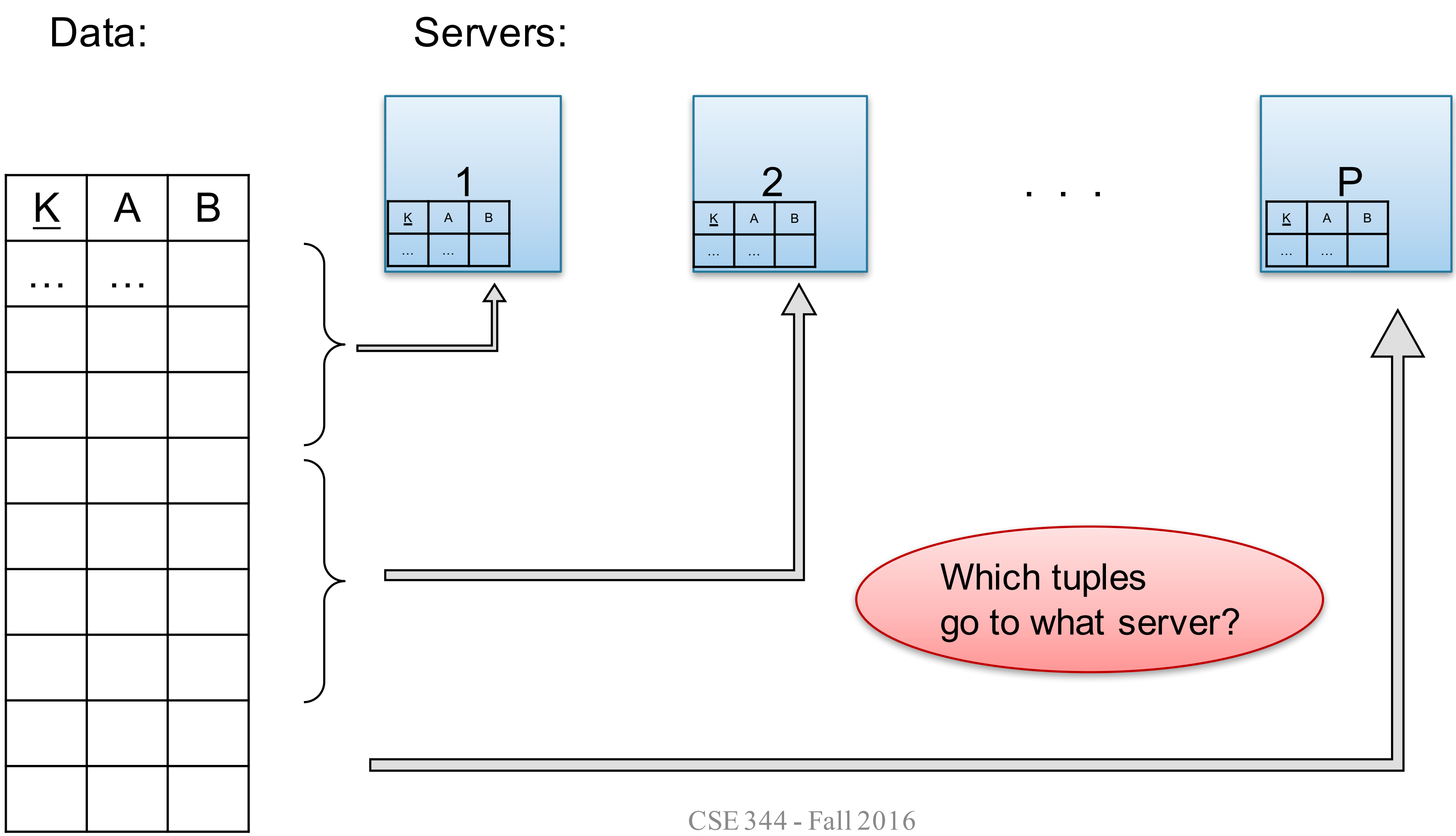
Horizontal Data Partitioning

Data:

K	A	B
…	…	

Servers:

Horizontal Data Partitioning



Horizontal Data Partitioning

- Block Partition:
  - Partition tuples arbitrarily s.t. $size(R_1) \approx \ldots \approx size(R_P)$
- Hash partitioned on attribute A:
  - Tuple t goes to chunk i, where i = h(t.A) mod P + 1
- Range partitioned on attribute A:
  - Partition the range of A into $-\infty = v_0 < v_1 < \ldots < v_P = \infty$
  - Tuple t goes to chunk i, if $v_{i-1} < t.A < v_i$

Parallel Group By

Data: R(<u>K</u>,A,B,C)
Query: $\gamma_{A,sum(C)}(R)$

How to compute if:

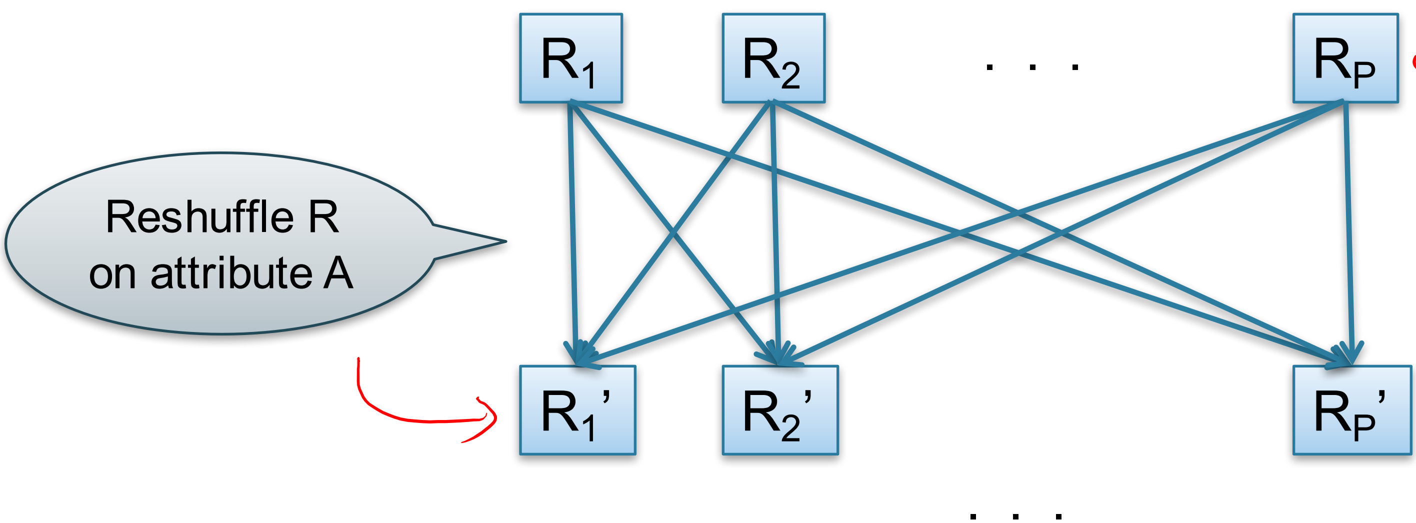
- R is hash-partitioned on A
- R is block-partitioned
- R is hash-partitioned on K

Parallel Group By

Data: R(K,A,B,C)

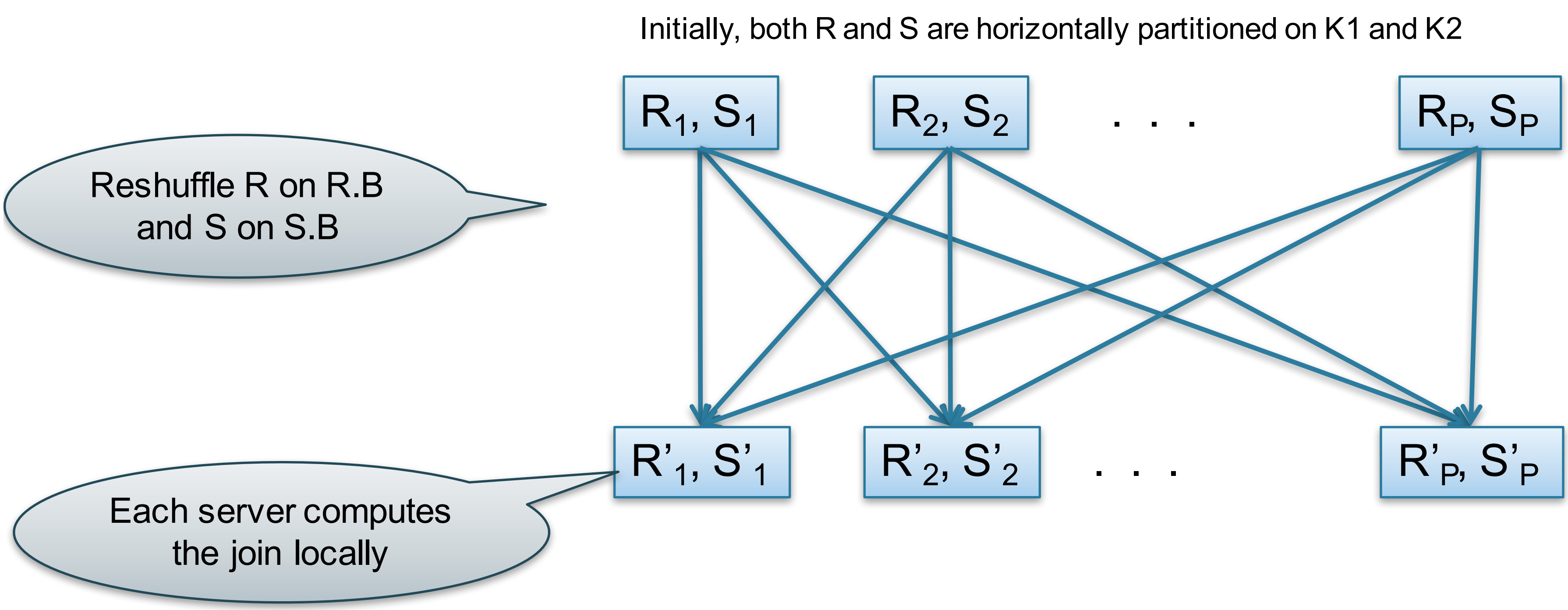
Query: $\gamma_{A,sum(C)}(R)$

- R is block-partitioned or hash-partitioned on K



Parallel Join

- Data: R(<u>K1</u>,A, B), S(<u>K2</u>, B, C)
- Query: R(<u>K1</u>,A,B) $\bowtie$ S(<u>K2</u>,B,C)



Data: R(K1,A, B), S(K2, B, C)

Query: R(K1,A,B) ⋈ S(K2,B,C)

Partition

M1

R1

K1	B
1	20
2	50

S1

K2	B
101	50
102	50

M2

R2

K1	B
3	20
4	20

S2

K2	B
201	20
202	50

Shuffle

Local Join

M1

R1'

K1	B
1	20
3	20
4	20

⋈

S1'

K2	B
201	20

M2

R2'

K1	B
2	50

⋈

S2'

K2	B
101	50
102	50
202	50

Speedup and Scaleup

- Consider:
  - Query: $\gamma_{A,sum(C)}(R)$
  - Runtime: dominated by reading chunks from disk
- If we double the number of nodes P, what is the new running time?
  - Half (each server holds ½ as many chunks)
- If we double both P and the size of R, what is the new running time?
  - Same (each server holds the same # of chunks)

Uniform Data v.s. Skewed Data

- Let R(K,A,B,C); which of the following partition methods may result in skewed partitions?

- Block partition

Uniform

- Hash-partition
  - On the key K
  - On the attribute A

Uniform

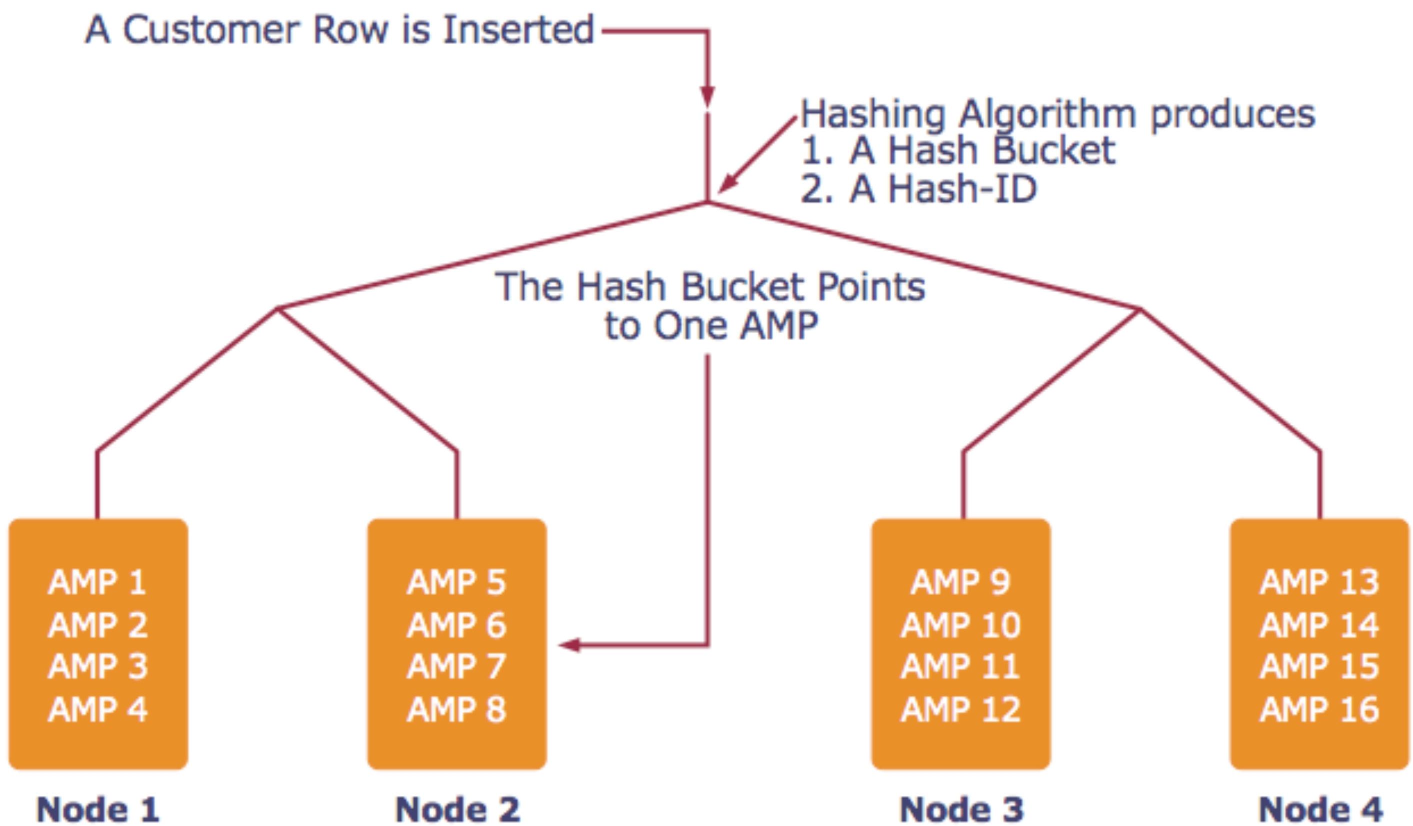
May be skewed

Assuming good hash function

E.g. when all records have the same value of the attribute A, then all records end up in the same partition

Loading Data into a Parallel DBMS

Example using Teradata

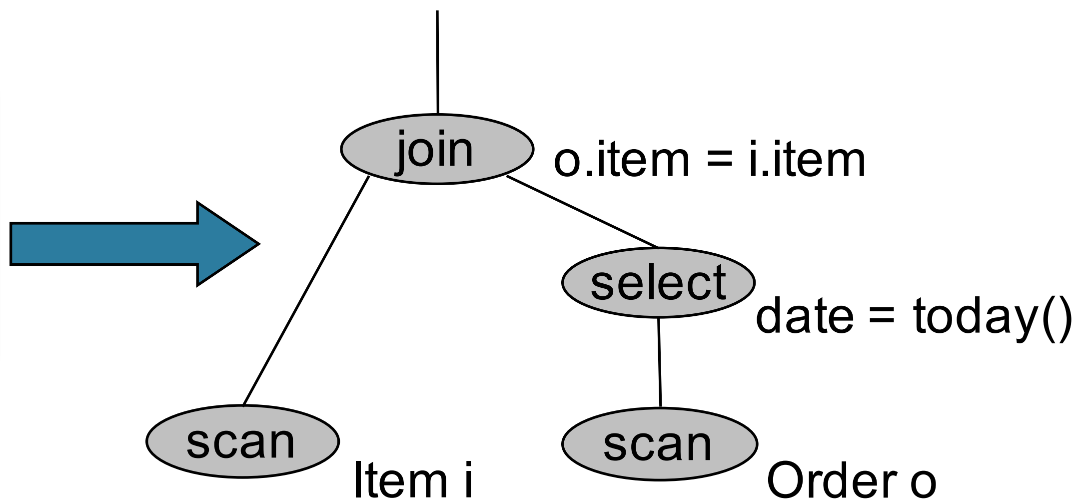


AMP = "Access Module Processor" = unit of parallelism

Example Parallel Query Execution

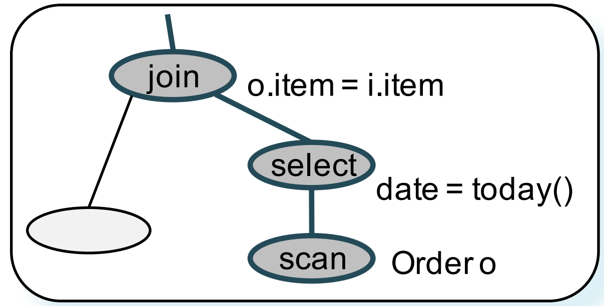
Find all orders from today, along with the items ordered

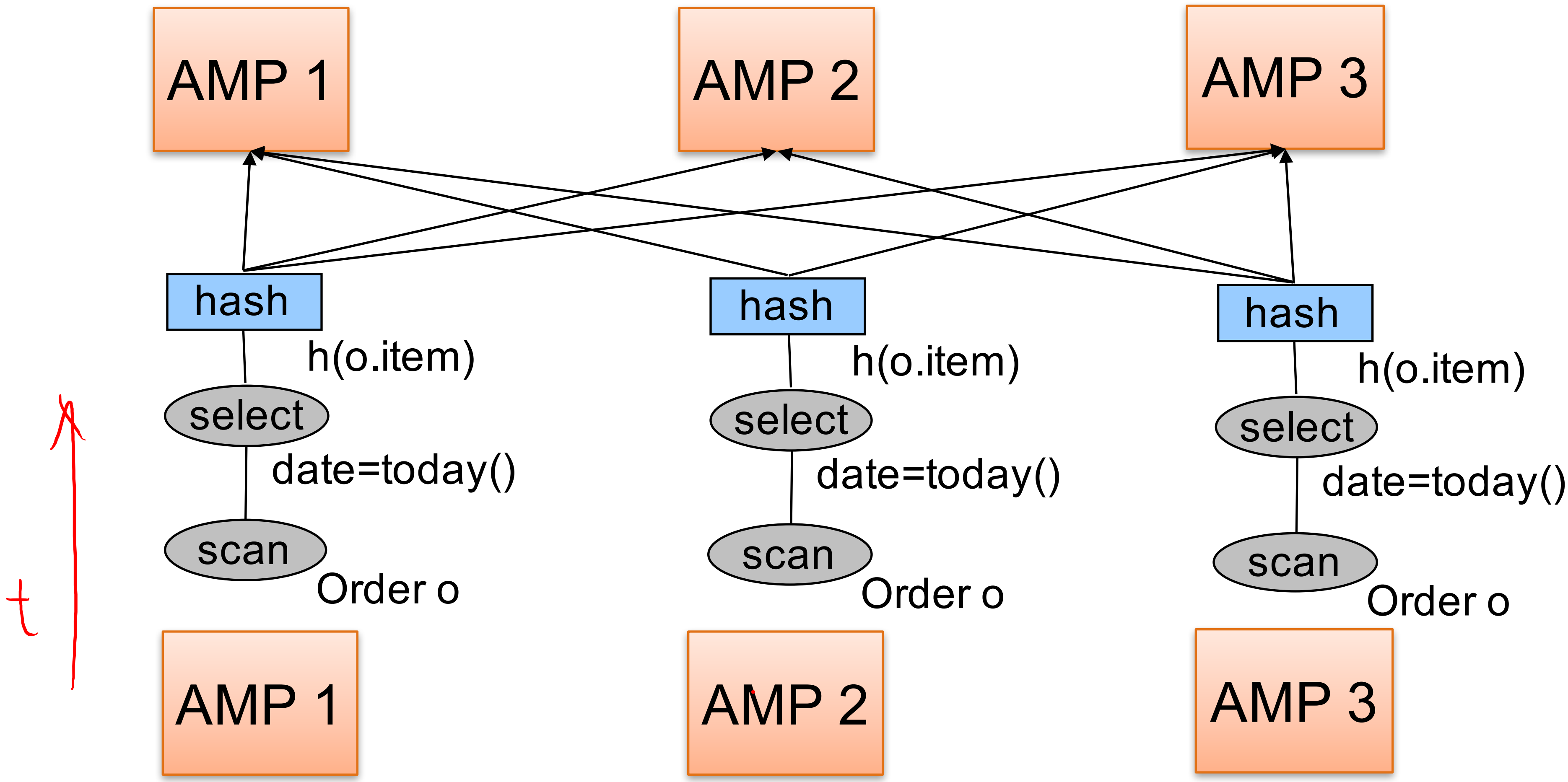
```
SELECT *
  FROM Order o, Line i
 WHERE o.item = i.item
   AND o.date = today()
```



Order(oid, item, date), Line(item, …)

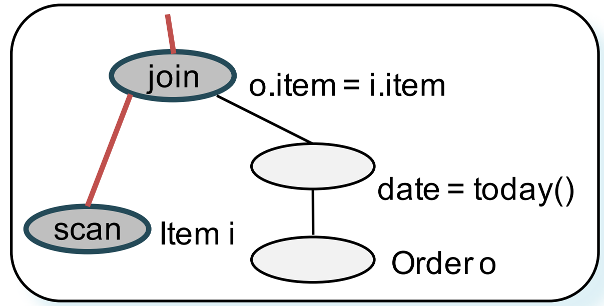
Example Parallel Query Execution

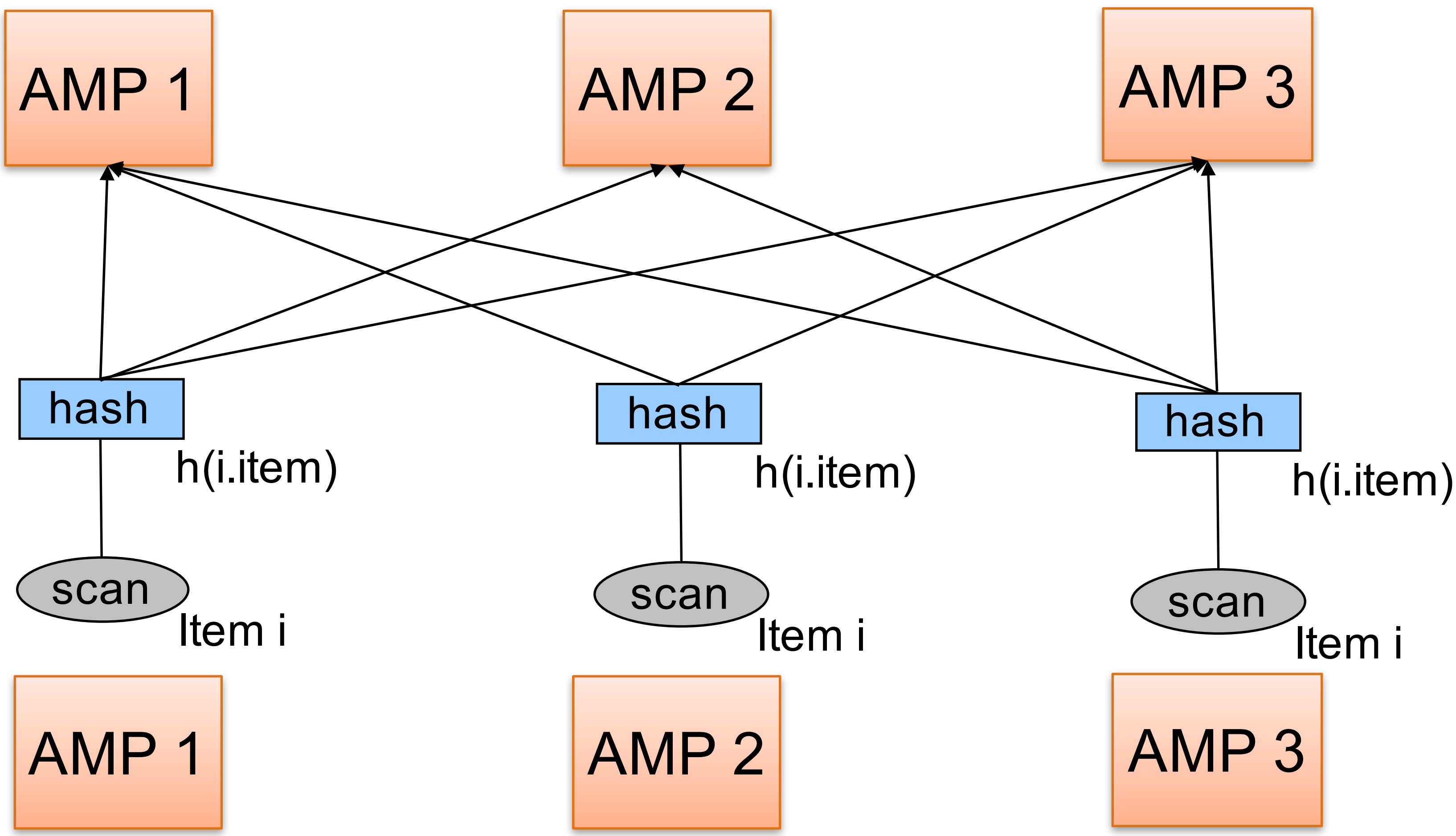




Order(oid, item, date), Line(item, …)

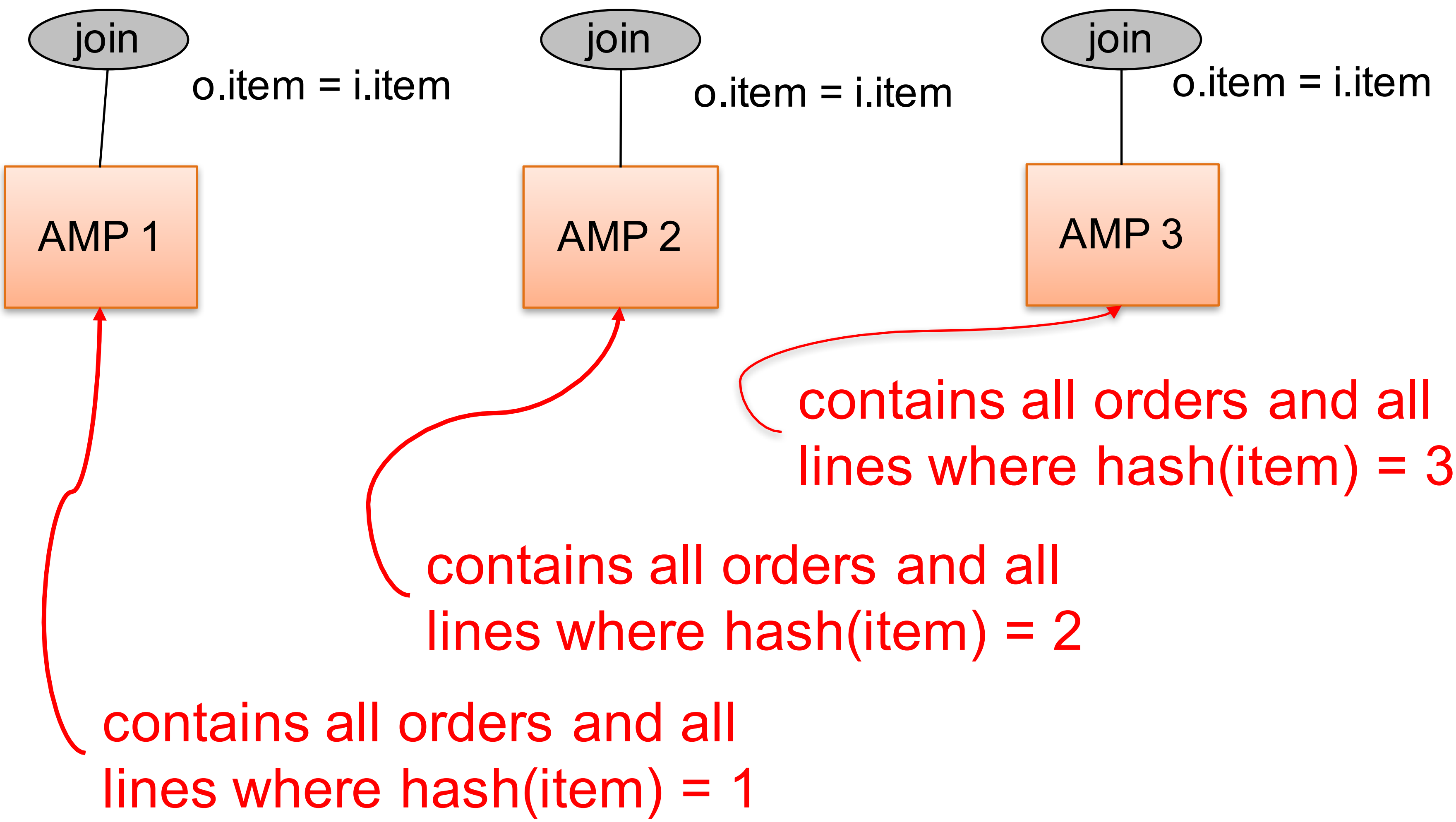
Example Parallel Query Execution





Order(oid, item, date), Line(item, …)

Example Parallel Query Execution



Parallel Data Processing @ 2000

Optional Reading

- Original paper: https://www.usenix.org/legacy/events/osdi04/tech/dean.html
- Rebuttal to a comparison with parallel DBs: http://dl.acm.org/citation.cfm?doid=1629175.1629198
- Chapter 2 (Sections 1,2,3 only) of Mining of Massive Datasets, by Rajaraman and Ullman http://i.stanford.edu/~ullman/mmds.html

Distributed File System (DFS)

- For very large files: TBs, PBs
- Each file is partitioned into *chunks*, typically 64MB
- Each chunk is replicated several times (≥3), on different racks, for fault tolerance
- Implementations:
  - Google's DFS: GFS, proprietary
  - Hadoop's DFS: HDFS, open source

MapReduce

- Google: paper published 2004
- Free variant: Hadoop

- MapReduce = high-level programming model and implementation for large-scale parallel data processing

Typical Problems Solved by MR

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, transform
- Write the results

Paradigm stays the same, change map and reduce functions for different problems

slide source: Jeff Dean

Map Reduce Data Model

Instance: Files!

- where a file = a bag of `(key, value)` pairs

Schema: None!

- just like other key-value data models

Query language: a MapReduce program:

- Input: a bag of `(inputkey, value)` pairs
- Output: a bag of `(outputkey, value)` pairs

Step 1: the MAP Phase

User provides the MAP-function:

- Input: `(input key, value)`
- Output: bag of `(intermediate key, value)`

System applies the map function in parallel to all `(input key, value)` pairs in the input file

Step 2: the REDUCE Phase

User provides the REDUCE function:

- Input: `(intermediate key, bag of values)`
- Output: bag of output `(values)`

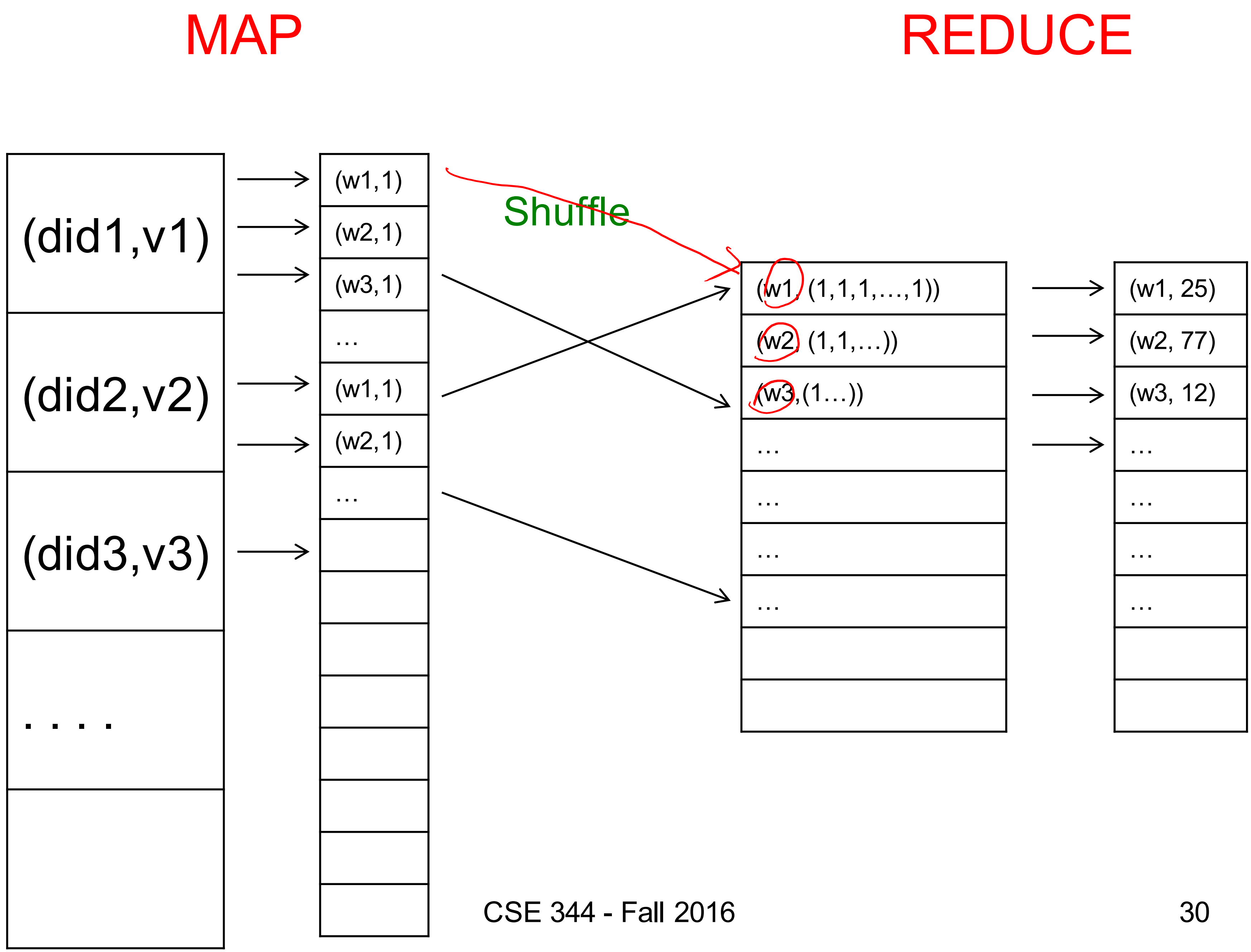
System groups all pairs with the same intermediate key, and passes the bag of values to the REDUCE function

Example

- Counting the number of occurrences of each word in a large collection of documents
- Each Document
  - The key = document id (did)
  - The value = set of words (word)

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```


MAP
REDUCE
(did1,v1)
(did2,v2)
(did3,v3)
. . . . .
(w1,1)
(w2,1)
(w3,1)
…
(w1,1)
(w2,1)
…
Shuffle
(w1, (1,1,1,…,1))
(w2, (1,1,…))
(w3,(1…))
…
…
…
…
(w1, 25)
(w2, 77)
(w3, 12)
…
…
…
…

Jobs v.s. Tasks

- A MapReduce Job
  - One single “query,” e.g., count the words in all docs
  - More complex queries may consists of multiple jobs
- A Map Task, or a Reduce Task
  - A group of instantiations of the map-, or reduce-function, which are scheduled on a single worker